In the Specification:

Please amend the title of the application, which is located at page 1 as indicated below:

ADMINISTERING DEVICES WITH DOMAIN STATE OBJECTS BY CREATING, IN A FIRST DOMAIN, A DOMAIN STATE OBJECT, AND TRANSMITTING THE DOMAIN STATE OBJECT FROM THE FIRST DOMAIN TO A SECOND DOMAIN

Please amend the title of the abstract, which is located at page 100 as indicated below:

ADMINISTERING DEVICES WITH DOMAIN STATE OBJECTS BY CREATING, IN A FIRST DOMAIN, A DOMAIN STATE OBJECT, AND TRANSMITTING THE DOMAIN STATE OBJECT FROM THE FIRST DOMAIN TO A SECOND DOMAIN

Please amend the specification in the Definitions section by changing the paragraph beginning at page 12, line 1, as indicated below:

> The HomePlug protocol allows HomePlug-enabled devices to communicate across powerlines using Radio Frequency signals (RF). The HomePlug protocol uses Orthogonal Frequency Division Multiplexing (OFDM) to split the RF signal into multiple smaller sub-signals that are then transmitted from one HomePlug enabled-device to another HomePlug-enabled device at different frequencies across the powerline.

Please amend the specification in the section entitled 'Exemplary Architecture' by changing the paragraph beginning at page 18, line 24, as indicated below:

> In the exemplary architecture of Figure 1, the services gateway (1~~2~~06) includes a services framework (126). The services framework (126) of Figure 1 is a hosting platform for running 'services.' Services are the main building blocks for creating applications in the OSGi. An OSGi services framework (126) is written in Java and therefore, typically runs on a Java Virtual Machine (JVM) (150).

Please amend the specification in the section entitled 'Exemplary Architecture' by changing the paragraph beginning at page 24, line 6, as indicated below:

> OSGi Stands for 'Open Services Gateway Initiative.' The OSGi specification is a Java-based application layer framework that provides vendor neutral application and device layer APIs and functions for various devices using arbitrary communication protocols operating in networks in homes, cars, and other environments. OSGi works with a variety of networking technologies like Ethernet, Bluetooth, the 'Home, Audio and Video Interoperability standard' (HAVi), IEEE 1394, Universal Serial Bus (USB), WAP, X-10, Lon Works, HomePlug and various other networking technologies. The OSGi specification is available for free download from the OSGi website ~~at www.osgi.org~~.

Please amend the specification in the section entitled 'Exemplary Architecture' by changing the paragraph beginning at page 24, line 16, as indicated below:

> The services gateway (1~~3~~06) of Figure 2 connects to device (316) via interface (119), and includes a service framework (126). In many example embodiments the service framework is an OSGi service framework (126). An OSGi service framework (126) is written in Java and therefore,

typically runs on a Java Virtual Machine (JVM). In OSGi, the service framework (126) of Figure 1 is a hosting platform for running 'services' (124). The term 'service' or 'services' in this disclosure, depending on context, generally refers to OSGi-compliant services.

Please amend the specification in the section entitled 'Exemplary Classes and Class Cooperation' by changing the paragraph beginning at page 26, line 13, as indicated below:

> The class diagram of Figure 3 includes an exemplary DML class (202). An instance of the exemplary DML class (202) of Figure 3 provides member methods that carry out the steps useful in administering devices in accordance with the present invention. The exemplary DML class of Figure 3 is shown with an Activator.start() (557) method so that the DML can be started as a service in an OSGi framework. Although only one member method is shown for this DML, DMLs in fact will often have more member methods as needed for a particular embodiment. The DML class of Figure 3 also includes member data elements for storing references to services classes, often created by the DML's constructor. In this example, the DML provides storage fields for references to a metric service (552), a metric range service (558), a communication service (554), an action service (560), a device service (556), a metric vector service (559) and a metric space service (561), and dynamic action list service (563).

Please amend the specification in the section entitled 'Exemplary Classes and Class Cooperation' by changing the paragraph beginning at page 34, line 17, as indicated below:

> Objects of the exemplary metric vector class also typically include member methods for determining if the metric vector is outside a user metric space, and a start () method (495). This exemplary metric vector class is an example of a class that can in various embodiments be used as a generic class, instances of which can be used to store or represent more than one type of vector having identical or similar member data elements. Alternatively in other embodiments, a class such as this example metric

vector class can be used as a base class to be extended by concrete derived classes each of which can have disparate member data types.

Please amend the specification in the section entitled 'Exemplary Classes and Class Cooperation' by changing the paragraph beginning at page 36, line 9, as indicated below:

The class diagram of Figure 3 includes a metric space service class (209). The metric space service class (209) includes a member method createMetricSpace() (567) that searches a metric space list, or other data structure, to identify a metric space for a user. If no such metric space exists, createMetricSpace() instantiates one and stores the metric space ID in the metric space list. Creating a metric space object can be implemented by way of the following exemplary psuedocode:

Please amend the specification in the section entitled 'Exemplary Classes and Class Cooperation' by changing the paragraph beginning at page 39, line 10, as indicated below:

The createActionList () method (564) in ActionService class instantiates a metric action list for a user metric with "ActionList anActionList = new ActionList()." CreateActionList() then searches an action record table in a database for records having user IDs and metric IDs matching its call parameters. For each matching record in the table, createActionList() instantiates an action object through its switch statement. The switch statement selects a particular concrete derived action class for each action ID retrieved from the action record table. CreateActionList() stores a references to each action object in the action list with "anActionList.add()." CreateActionList() returns a reference to the action list with "return anActionList."

Please amend the specification in the section entitled 'Exemplary Classes and Class Cooperation' by changing the paragraph beginning at page 39, line 20, as indicated below:

> The class diagram of Figure 3 includes an exemplary action class (216). An instance of the action class represents an action that when executed results in the administration of a device. The exemplary action class of Figure 3 includes an action ID field (450), an ActionDescription field (317), and a createDeviceStateObject () method (911). The doAction() method (456) in the exemplary action class (216) is programmed to obtain a device list (458) from, for example, a call to DeviceService.createDeviceList(). Action.doAction() (456) typically then also is programmed to call interface methods in each device in its device list to carry out the device controlling action.

Please amend the specification in the section entitled 'Administering Devices in Dependence Upon User Metrics' by changing the paragraph beginning at page 50, line 16, as indicated below:

> In many embodiments of the method of Figure 6, receiving (302) a user metric includes receiving a user metric from a metric sensor (406). In some examples of the method of Figure 6, the metric sensor (406) reads an indication of ~~a user's condition~~ the condition of a user (300), creates a user metric in dependence upon the indication of a user's condition, and transmits the user metric to a DML. In many embodiments, the metric sensor transmits the user metric to the DML in a predefined data structure, such as the metric (206) of Figure 6, to the DML using, for example, protocols such as Bluetooth, 802.11, HTTP, WAP, or any other protocol that will occur to those of skill in the art.

Please amend the specification in the section entitled 'Administering Devices in Dependence Upon User Metrics' by changing the paragraph beginning at page 59, line 1, as indicated below:

> In the method of Figure 8, identifying (310) an action in dependence upon the user metric includes identifying (512) an action in dependence upon the degree (504) to which the value of the user metric (206) is outside (309) the metric range and also often in dependence upon the direction in which the metric is out of range. In many embodiments of the method of Figure 8, identifying (512) the action in dependence upon the degree (504) to which the user metric is outside the predefined metric range includes retrieving an action ID from an ~~metric~~ action list (~~622~~313) organized by metric ID, user ID, degree, and direction.

Please amend the specification in the section entitled 'Administering Devices With Domain State Objects' by changing the paragraph beginning at page 77, line 19, as indicated below:

> The method of Figure 13 includes transmitting (916) the domain state object (914) from the first domain (912) to a second domain (918). A second domain means any particular networked environment other than the first domain. That is, another networked environment. Typical second domains, such as the second domain of Figure 13, include a second domain DML (904). The second domain DML includes application programming useful in implementing methods of administering devices in accordance with the present invention.

Please amend the specification in the section entitled 'Administering Devices With Domain State Objects' by changing the paragraph beginning at page 84, line 21, as indicated below:

> One advantage to downloading the address of the domain state object to the mobile sensor rather than the downloading the domain state object itself is that less memory is required on the mobile device. However, by downloading only the address, the second domain must be coupled for data communications, by a network (906), with the address where the domain state object is located. The address may be located on in the first domain, on an ISP, or any other location that will occur to those of skill in the art.

Please amend the specification in the section entitled 'Administering Devices With Domain State Objects' by changing the paragraph beginning at page 86, line 7, as indicated below:

In the method of Figure 17, selecting (968) an action ID (315) in dependence upon the current device state object (926) includes identifying (967) a device in the second domain that is similar to a device included in the current device state object (926), that is, a similar second domain device (969). In many examples of the method of Figure 13, a device in the second domain that is 'similar' to a device included in the current device state object is a device that affects user condition in a manner similar to the device included in the current device state object. Two devices do not have to be the same type of devices to be similar. For example, a desk lamp may be similar to an overhead light, a fan may be similar to an air conditioner, radio may be similar to a CD player. Each of these devices differ in operation, however, each may affect user condition similarly. Furthermore, in some examples of the method of Figure 17, a plurality of devices may be similar to a single device. For example, a car heater and car seat heaters together may operate to affect user condition in a manner similar to a heater in a home.